Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

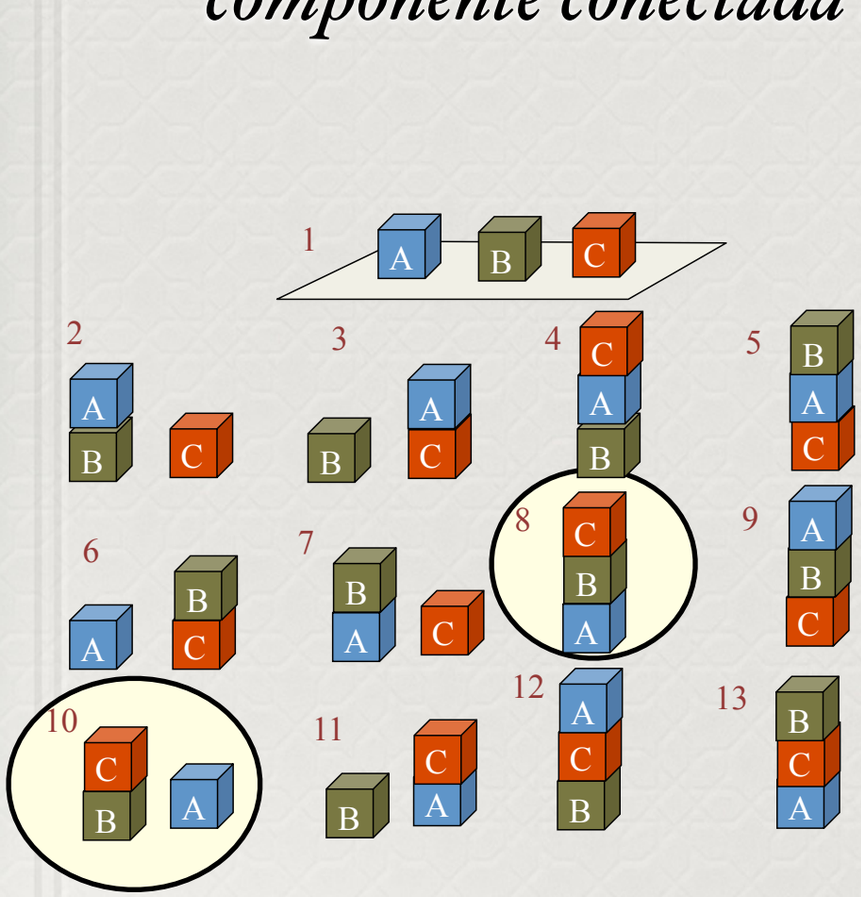
AULA 14 - AI PLANNING ALGORITHMS

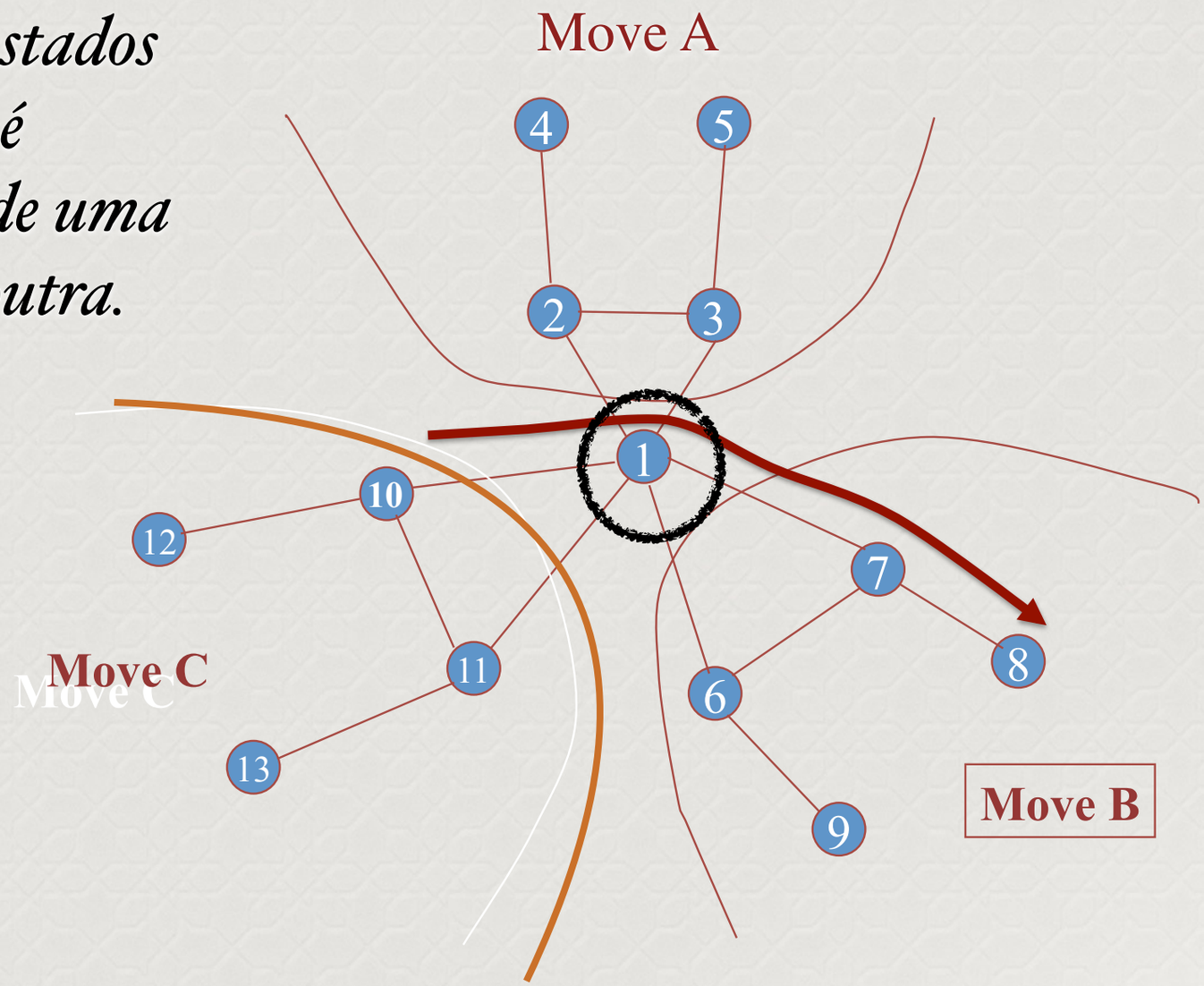
Prof. José Reinaldo Silva

reinaldo@usp.br

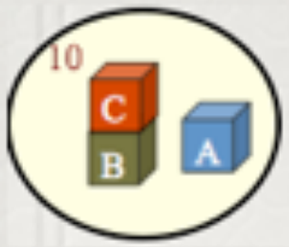
d-l@b

Uma análise do espaço de estados mostra que o estado 1 é "preferencial" para passar de uma componente conectada a outra.

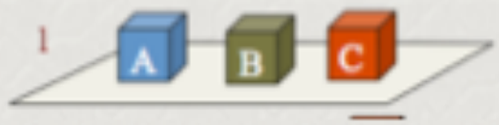




Representação dos "estados" em lógica: estado inicial e estado "preferencial.



sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)



sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)

estado inicial

estado objetivo

Conjunto de Ações

Problema de planejamento

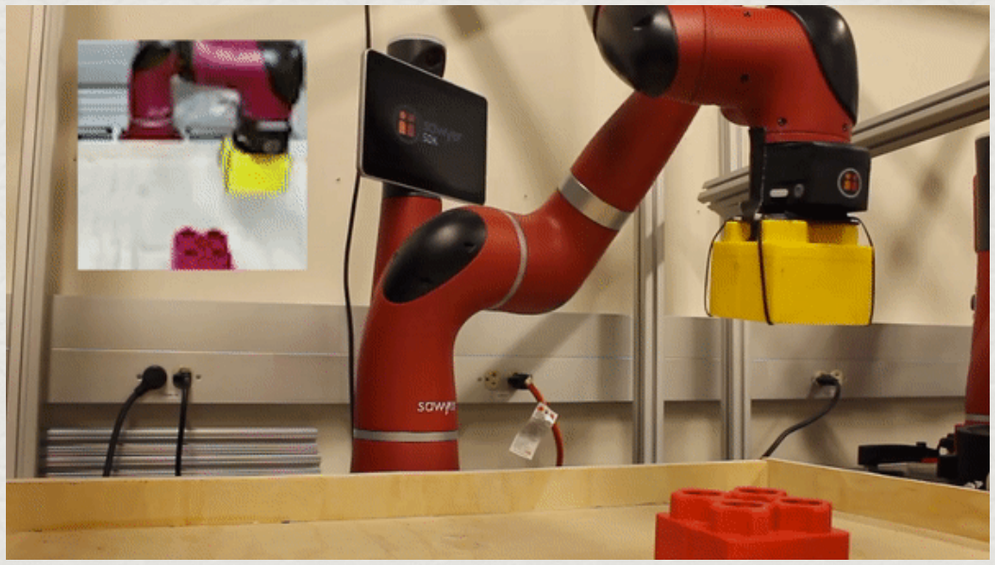
Critério de aplicação das ações

Percepção das mudanças de estado

Análise de custo

Base de conhecimento

O conjunto de ações é baseado nos movimentos de um braço robótico onde o efetuador (garra) só pode segurar um bloco de cada vez. A deposição só pode ser feita sobre um bloco livre, que não tenha nenhum outro bloco em cima dele.

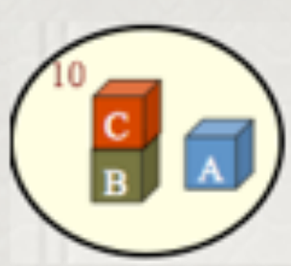


Uma ação desse tipo pode ser definida como $move(X, Y)$, onde um robô pode mover o bloco X para cima do bloco Y ou para a mesa.

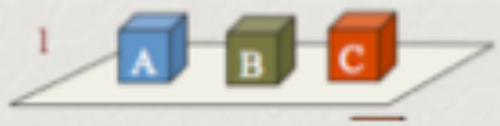
Pré-condições

Pós-condições

$move(c, mesa)$.



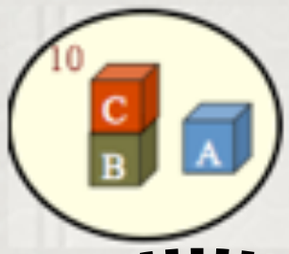
sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)

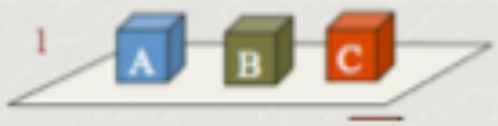


sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)

$$move(c, mesa)\ .$$

remove([sobre(c,b)], s) add([sobre(c,mesa), livre(b)], s)

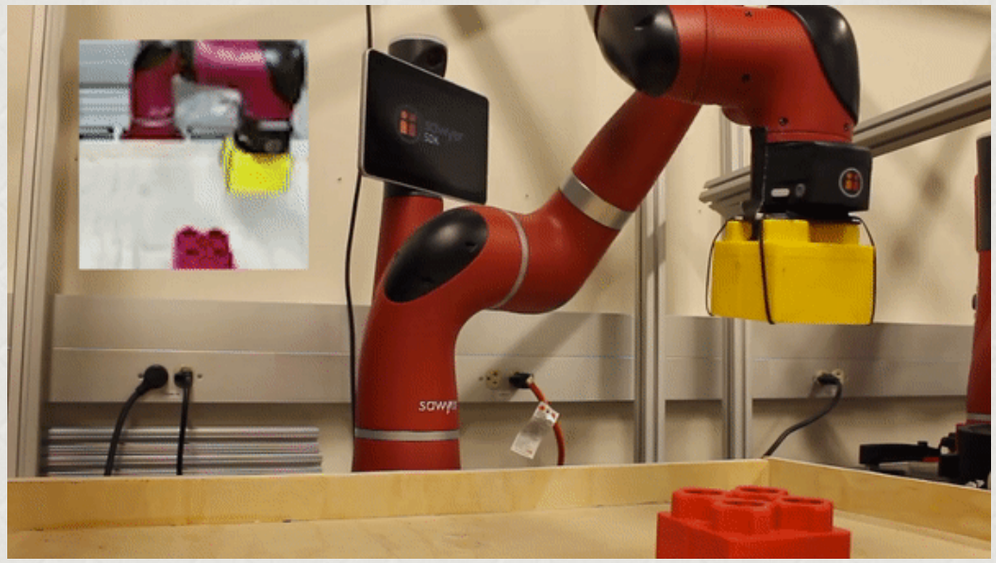




S

sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)

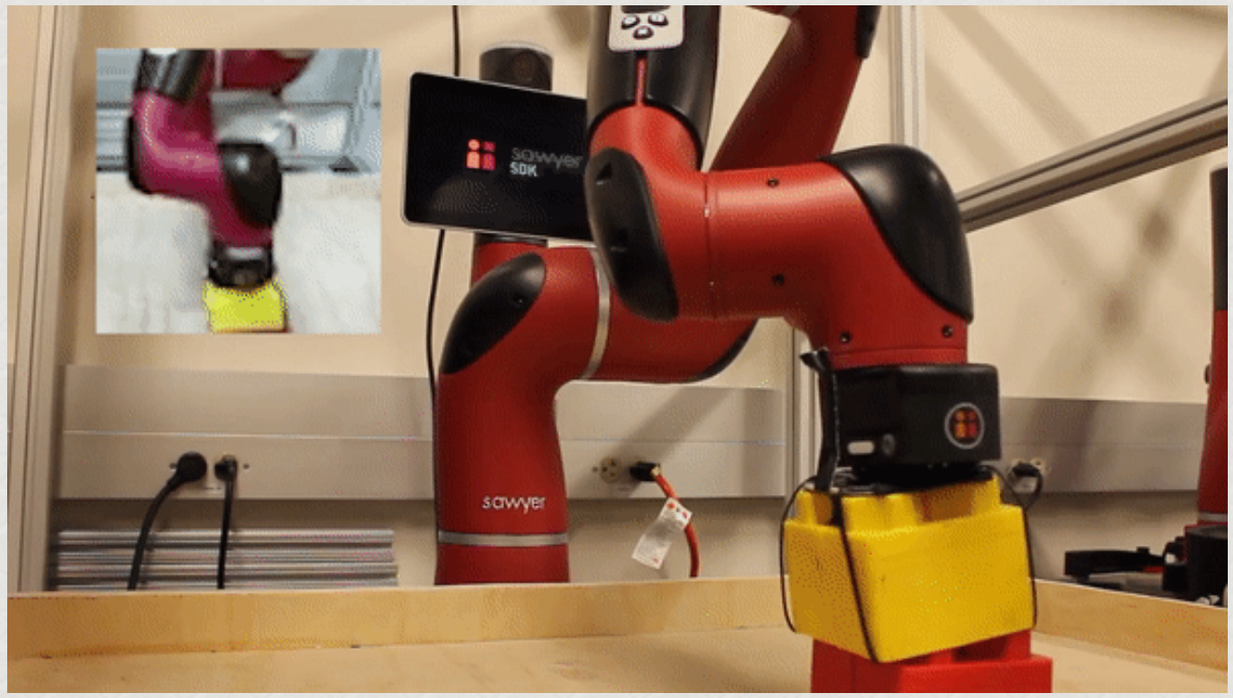
sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)



$$move(X,Y) :- livre(X), livre(Y), livre(garra), remove(livre(Y)), add(sobre(X,Y)).$$

Intuitivamente, as ações não são somente enumeradas mas definidas como predicados, com regras para sua aplicação. O evento é aplicado identificando uma lista de termos que deve ser removida da descrição do estado origem, e acrescentando uma nova lista a esse estado para obter o estado destino.

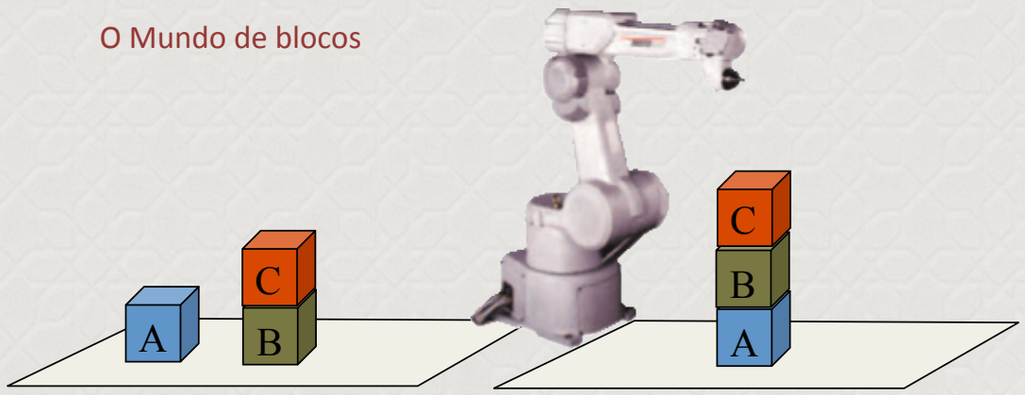
Será que isso pode ser generalizado como um método para resolver uma classe ampla de problemas de planejamento?.



Vamos discutir essa possibilidade e apresentar mais detalhadamente um algoritmo genérico para resolver problemas conhecido como STRIPS.

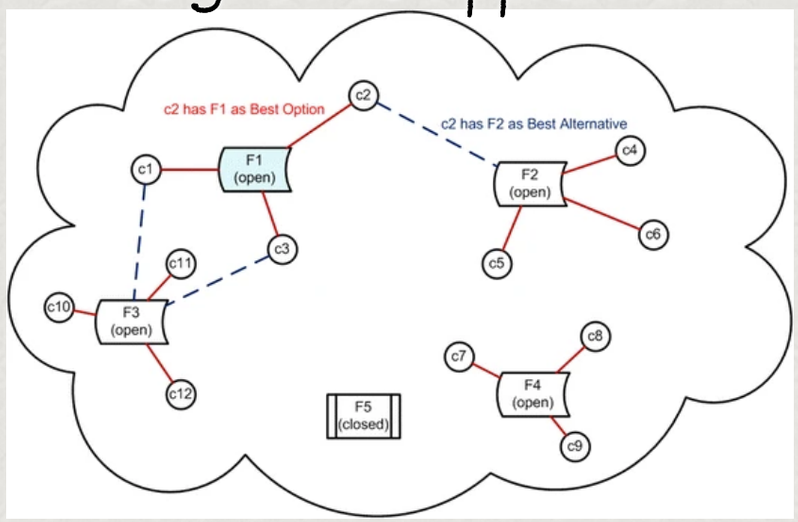
Vamos também começar a tratar dos aspectos práticos aplicados ao problema-modelo do mundo de blocos, que será o próximo exercício-programa.

O Mundo de blocos

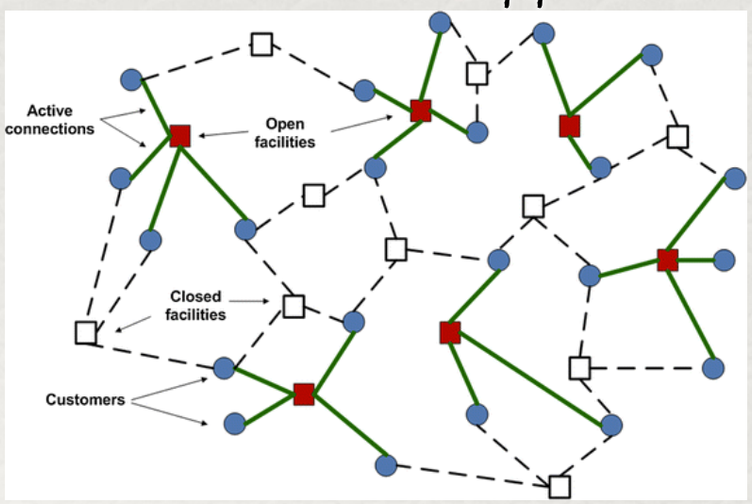


General Problem-solving approach

Cognitive approach



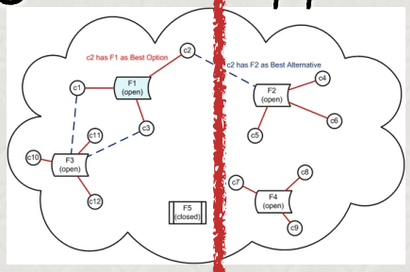
Deterministic approach



Stochastic and uncertainty approach

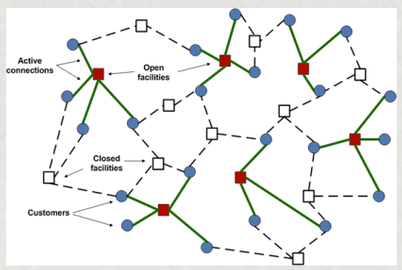
General Problem-solving approach

Cognitive approach



Modelos discretos, com heurísticas ou baseados em conhecimento (IA clássica).

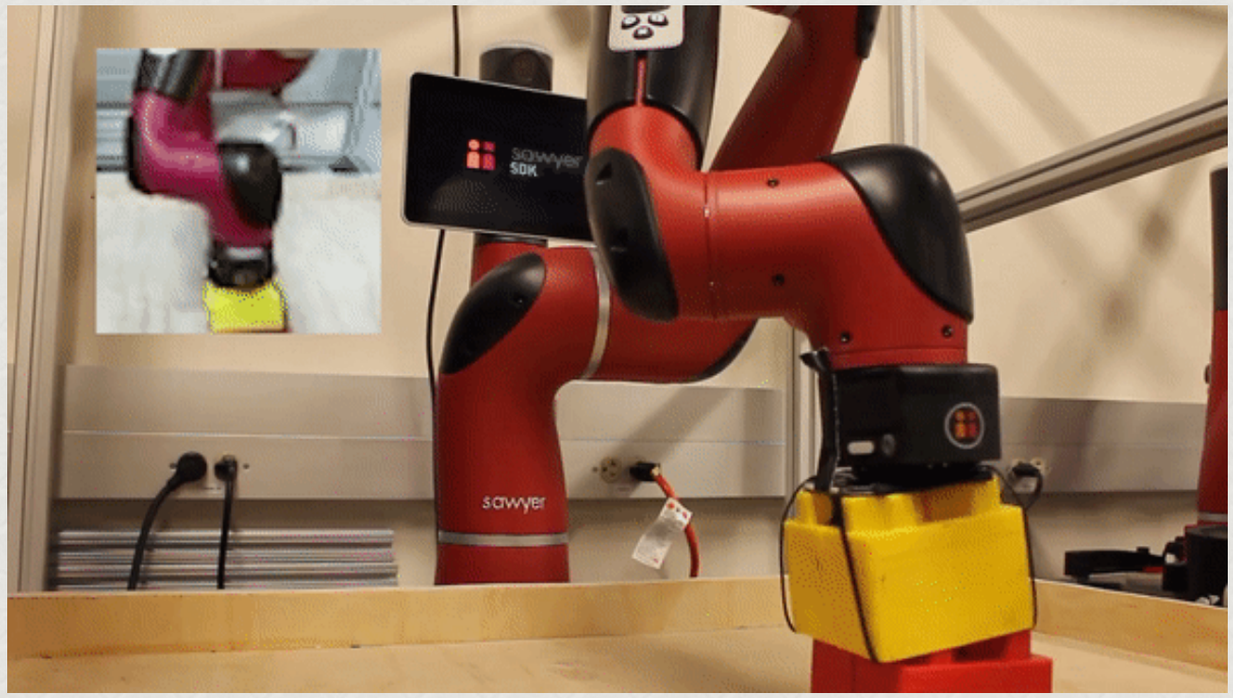
Deterministic approach



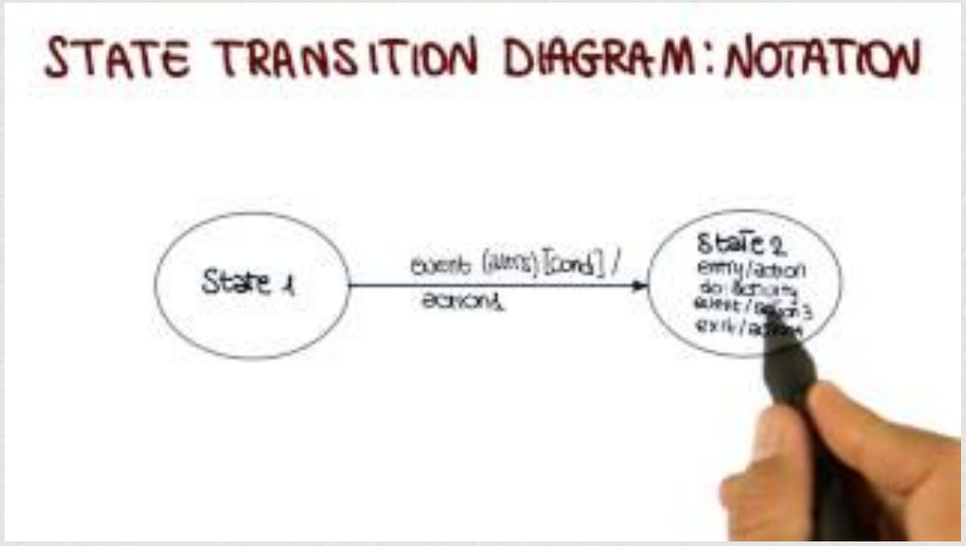
Modelos contínuos, discretos, computacionais.

Stochastic and uncertainty approach

Modelos discretos, estocásticos, baseados e bigdata.

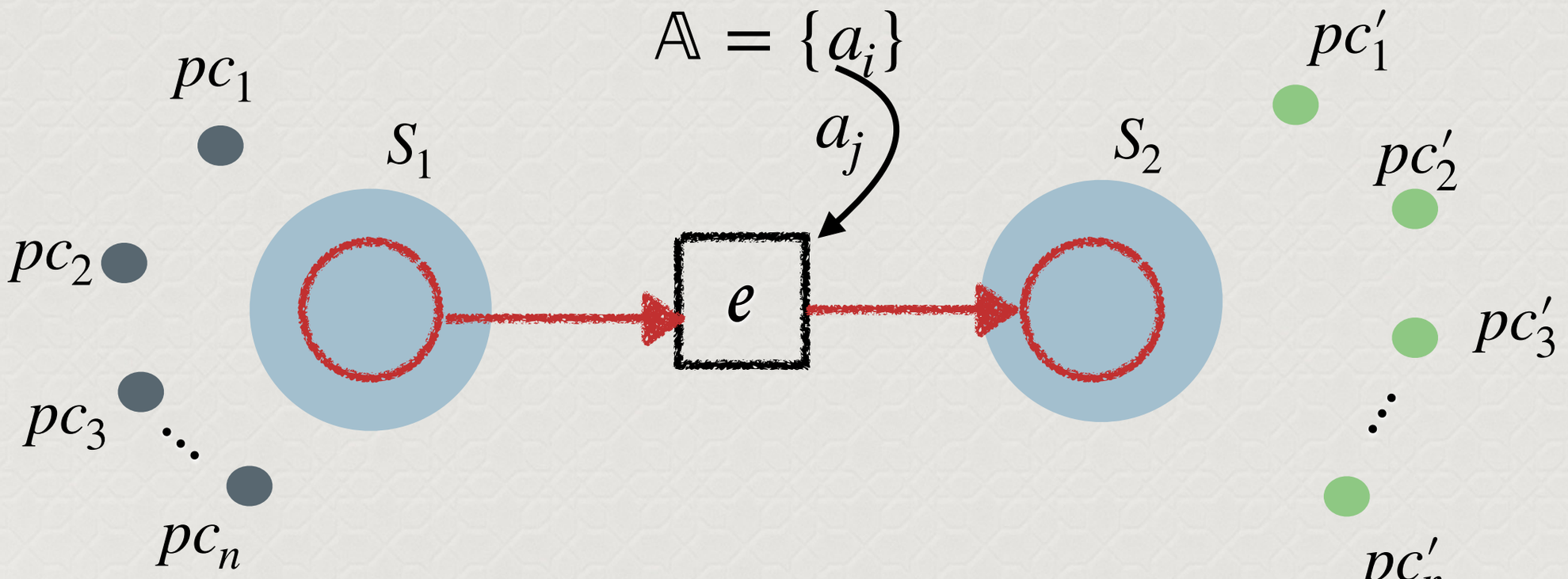


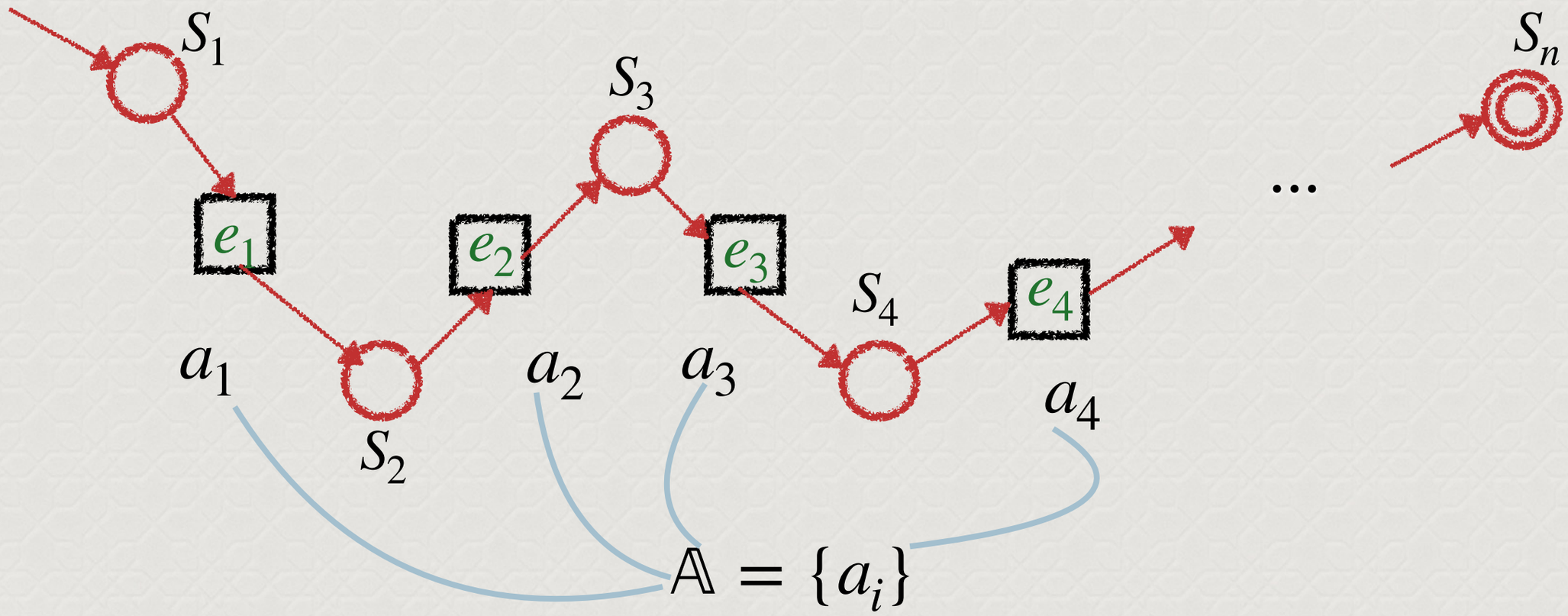
Continuaremos a discussão sobre a proposta do STRIPS, e chegaremos até um algoritmo genérico implementado sobre o mundo de blocos.



O planejamento clássico é fundamentado no paradigma estado-transição, assumindo que o "plano" ℘ será executado de forma discreta, uma ação por vez.

A cada instante, o sistema estará em um estado específico que "admite" ações que podem fazê-lo evoluir para um próximo estado.


$\mathbb{A} = \{a_i\}$
a_j
pc_1
pc_2
pc_3
pc_n
S_1
e
S_2
pc'_1
pc'_2
pc'_3
pc'_n



11.1 Definition of Classical Planning

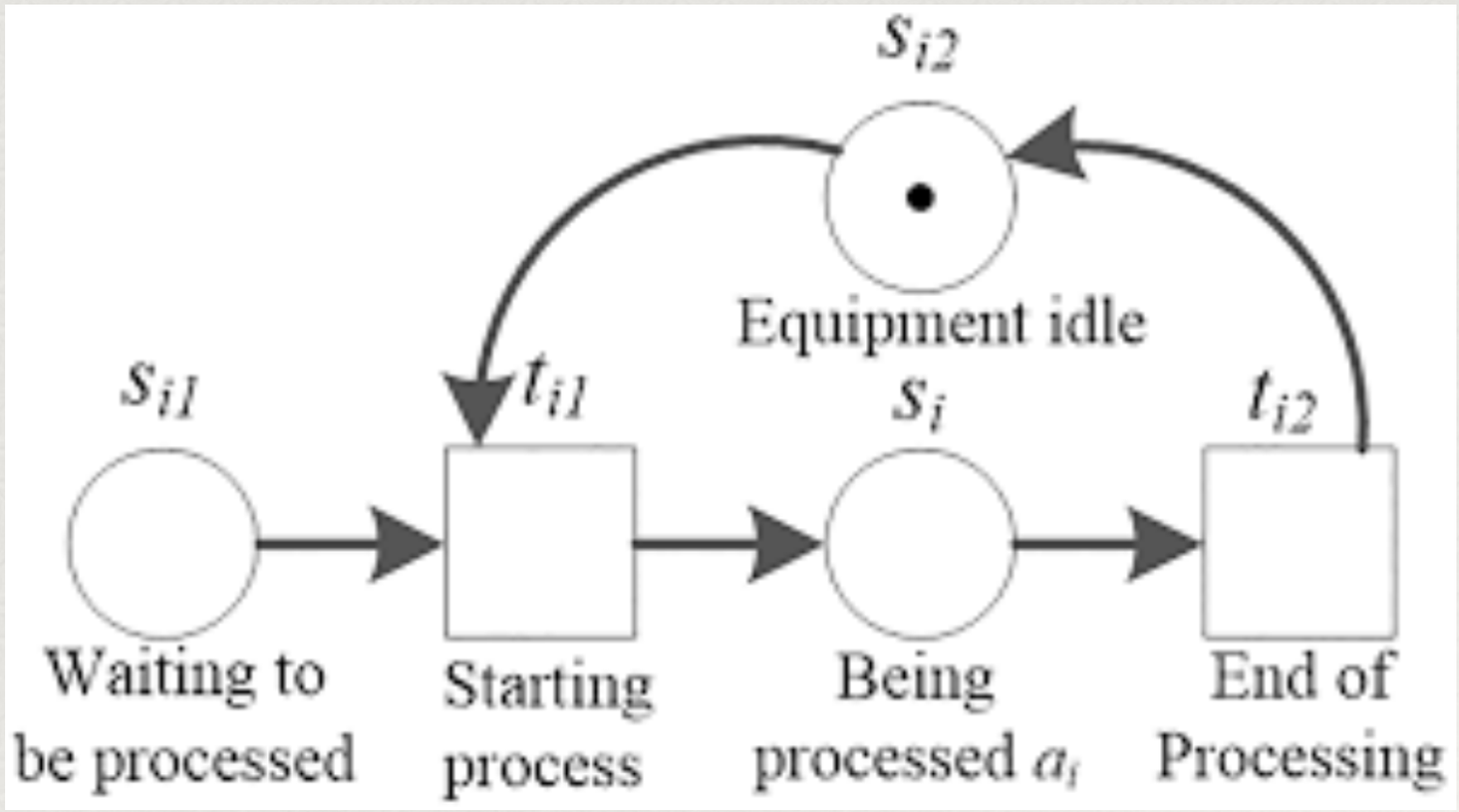
Classical planning

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment. We have seen two approaches to this task: the problem-solving agent of Chapter 3 and the hybrid propositional logical agent of Chapter 7. Both share two limitations. First, they both require ad hoc heuristics for each new domain: a heuristic evaluation function for search, and hand-written code for the hybrid wumpus agent. Second, they both need to explicitly represent an exponentially large state space. For example, in the propositional logic model of the wumpus world, the axiom for moving a step forward had to be repeated for all four agent orientations, T time steps, and n^2 current locations.

AI Magazine Volume 11 Number 2 (1990) (© AAAI)

AI Planning: Systems and Techniques[1]

James Hendler, Austin Tate, and Mark Drummond



General Planning approach

Domain independent

Knowledge Base

Domain dependent

O planejamento clássico é basicamente um processo de achar um arranjo de k ações pertencentes a um conjunto dado que leve o sistema do estado inicial ao estado final fornecido no "planning problem". Se todas as sequências de ações (sem repetições) fossem possíveis teríamos algo como k! possibilidades. Contando com as repetições e com o fato de que nem todas as sequências são válidas fisicamente, ainda assim o número de sequências cresce exponencialmente com o número de estados.

O método geral de resolução consiste em identificar como se configura estado (no caso por uma lista de predicados e constantes) e como se faz a "transição", modificando esta configuração de forma discreta (subtraindo alguns predicados e acrescentando outros).

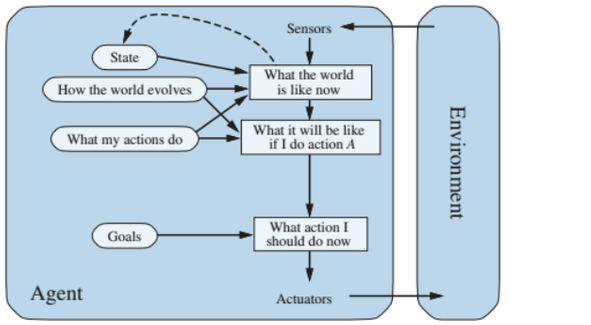


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Heurística ou regras de inferência

Existe um métodos geral, independente de domínio, para resolver problemas, em especial o problema de agentes com objetivo?.

STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving[1]

Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

Presented at the 2nd IJCAI, Imperial College, London, England, September 1–3, 1971.

ABSTRACT

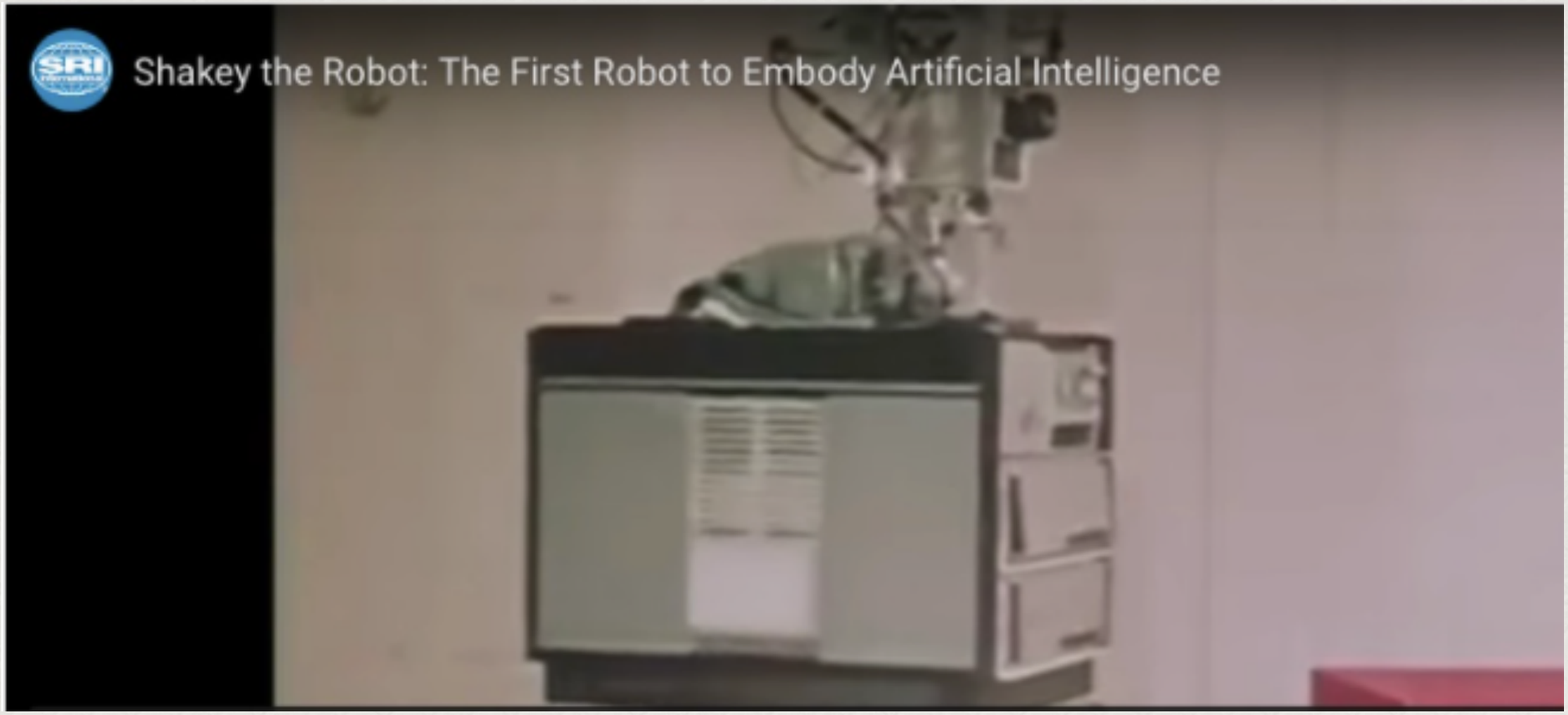
We describe a new problem solver called STRIPS that attempts to find a sequence of operators in a space of world models to transform a given initial world model into a model in which a given goal formula can be proven to be true. STRIPS represents a world model as an arbitrary collection of first-order predicate calculus formulas and is designed to work with models consisting of large numbers of formulas. It employs a resolution theorem prover to answer questions of particular models and uses means-ends analysis to guide it to the desired goal-satisfying model.

DESCRIPTIVE TERMS

Problem solving, theorem proving, robot planning, heuristic search.

1. Introduction

This paper describes a new problem-solving program called STRIPS (*ST*anford *R*esearch *I*nstitute *P*roblem *S*olver). An initial version of the program has been implemented in LISP on a PDP-10 and is being used in conjunction with robot research at SRI. STRIPS is a member of the class of


Shakey the Robot: The First Robot to Embody Artificial Intelligence

Lecture Notes for 6.824, Artificial Intelligence
 Rev. November, 1994

Graph Search and STRIPS Planning

1 Introduction to Graph Search

Consider the eight puzzle show in figure 1. This puzzle is played on a three by three square board containing eight tiles and an empty square. Any tile which is next to the empty square can be slid into the empty square leaving an opening (empty square) in the place that used to be occupied by the moved tile. Figure 1 shows how one state of the puzzle can be transformed into another configure by sliding a tile into the empty square. Given an initial state of the puzzle the objective is to find a sequence of legal moves that transform the initial state into some desired goal state. Figure 2 also shows a typical goal state.

The problem of finding a sequence of moves that transforms a given initial state into a given goal state can be formulated as a graph search problem. The nodes of the graph are the possible states of the puzzle and the arcs of the graph correspond to legal moves that transform one state into another. The problem is to find a path in this graph from a given initial state to a given goal state.

In the eight puzzle there are four types of legal moves. The empty square can move up, move down, move right, or move left. This allows the puzzle to be formulated in terms of four operations U, D, L, and R that move the empty square up, down, right and left respectively. If the empty square is already on the upper edge of the board then we define the operation U to

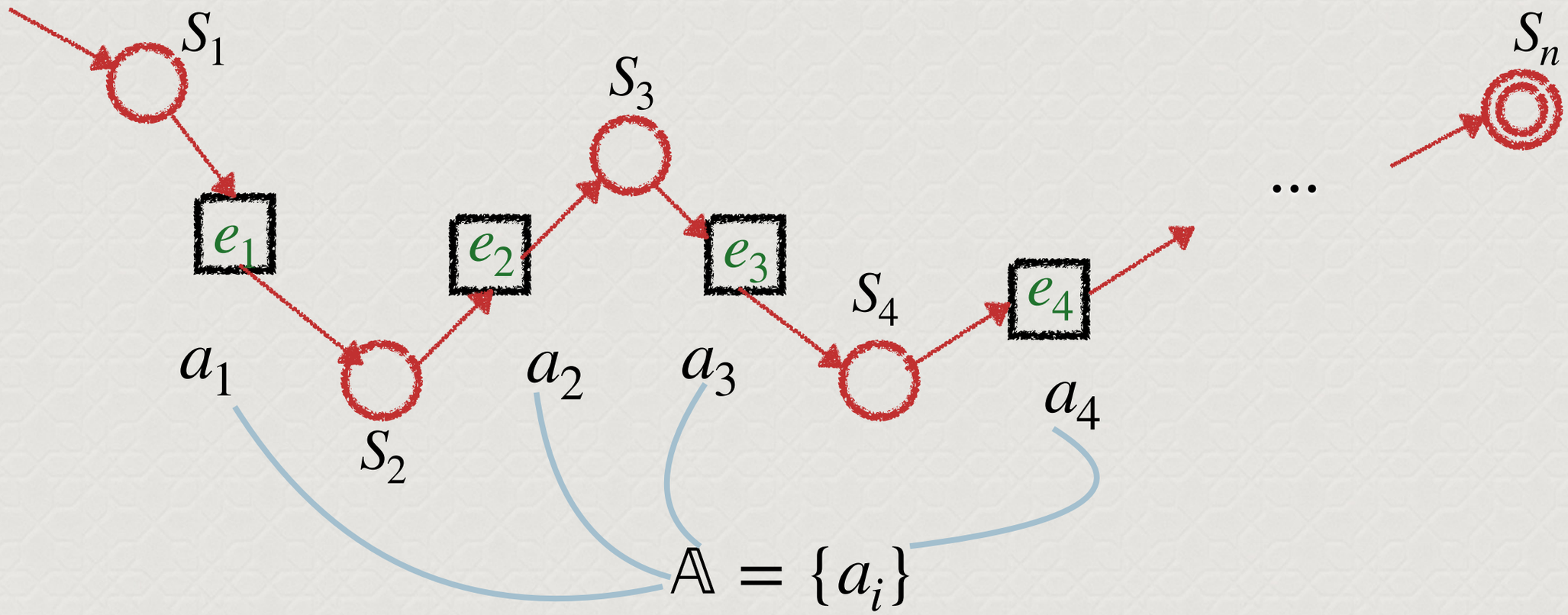
Definition: A *STRIPS planning problem* consists of a STRIPS operator specification, a set Σ of initial propositions and a set Ω of goal propositions.

Definition: A *solution* to a STRIPS planning problem is a plan (sequence of operators) α such that, in every graph where the STRIPS operator specification holds, we have $\Sigma \rightarrow [\alpha]\Omega$.

Definition: A *STRIPS operator specification* consists of a set of *operator symbols* where each operator symbol is associated with a *prerequisite list*, an *add list* and a *delete list* each of which is a set of proposition symbols.

Definition: A STRIPS operator specification is said to *hold* (or be *valid*) in a graph search problem if for each operator o_i, and each node n such that every prerequisite of o_i is true at n, we have the following conditions.

- All propositions on the add list of o_i are true at the node $o_i(n)$.
- If P is a proposition that is *not* on the delete list of o_i, and P is true at n, then P is true at $o_i(n)$.

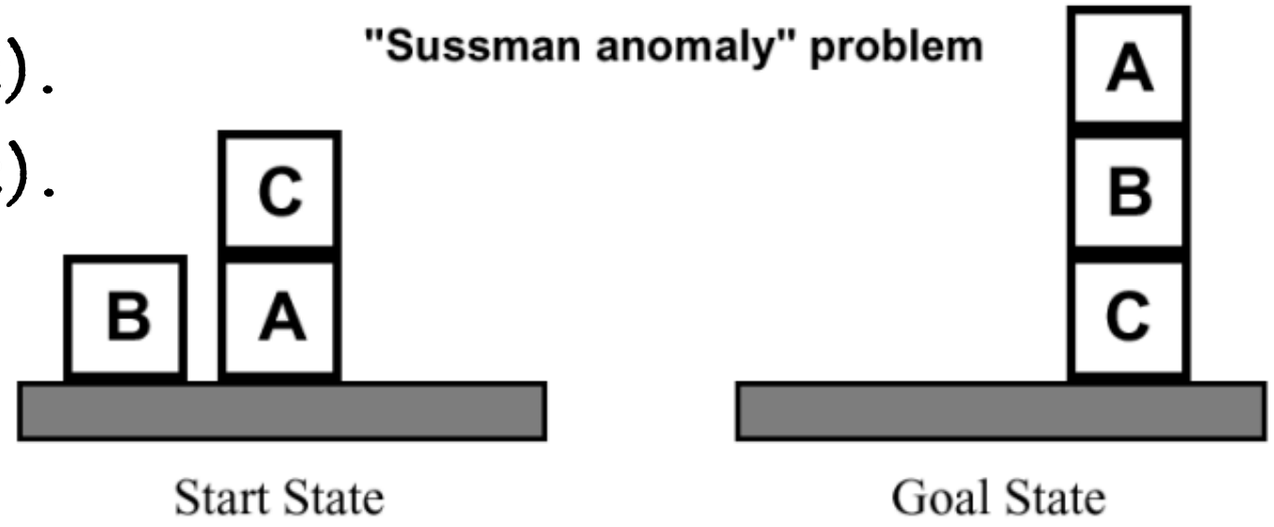


$$\mathbb{A} = \{a_i\}$$

- Apesar de geral o algoritmo não é completo, e pode gerar a anomalia de Sussman;
- Não há nenhuma garantia de convergência, portanto, a busca de solução pode divergir;
- O algoritmo não é otimizante.

Sussman Anomaly in the block world

on(b, table).
on(a, table).
on(c,a).
free(b).
free(c).



on(c, table).
on(a, b).
on(b,c).
free(a).

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

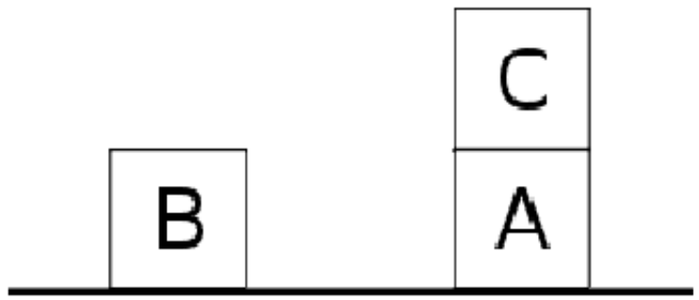
~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

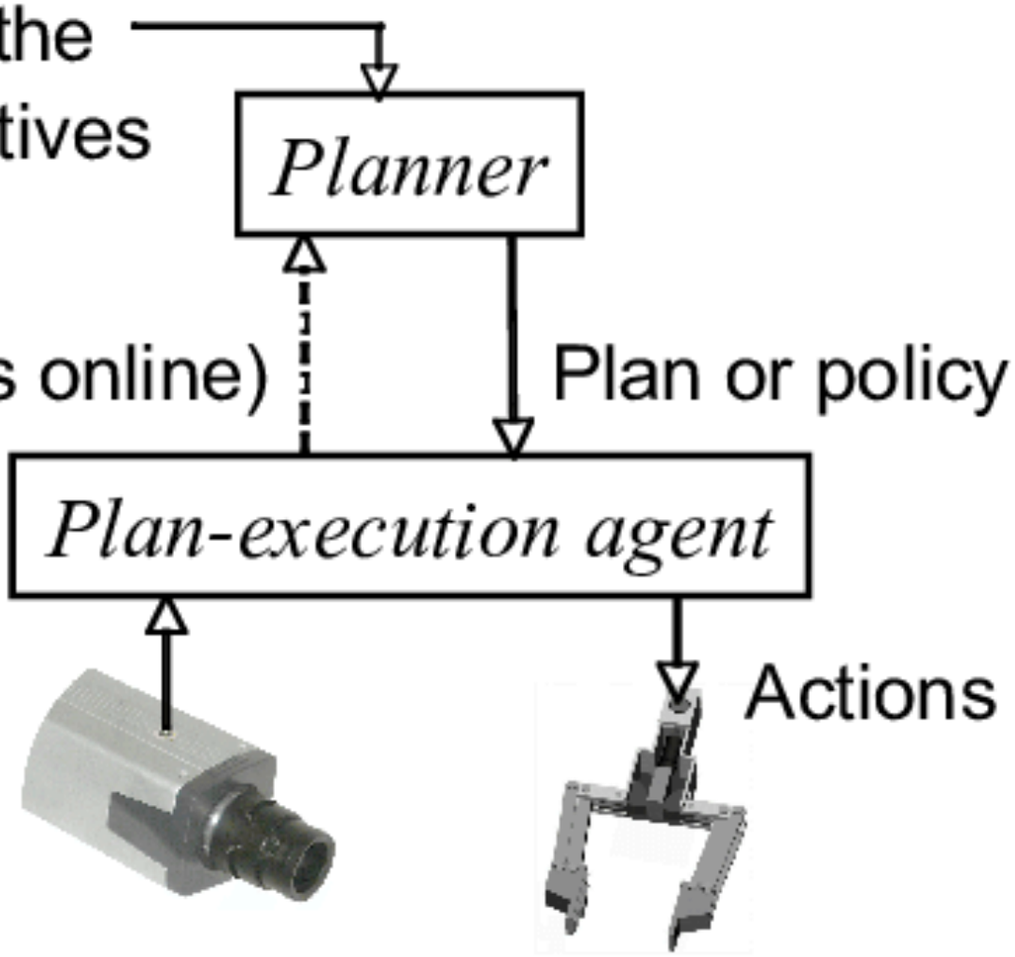
~On(x,z) Clear(z) On(x,Table)

Sussman Anomaly

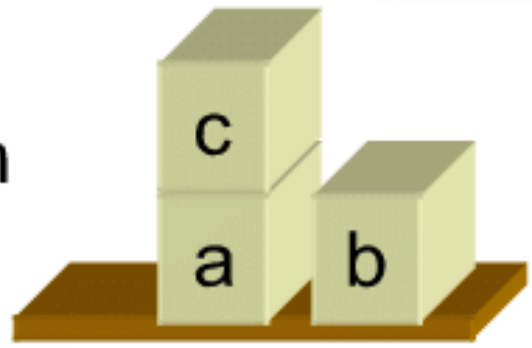


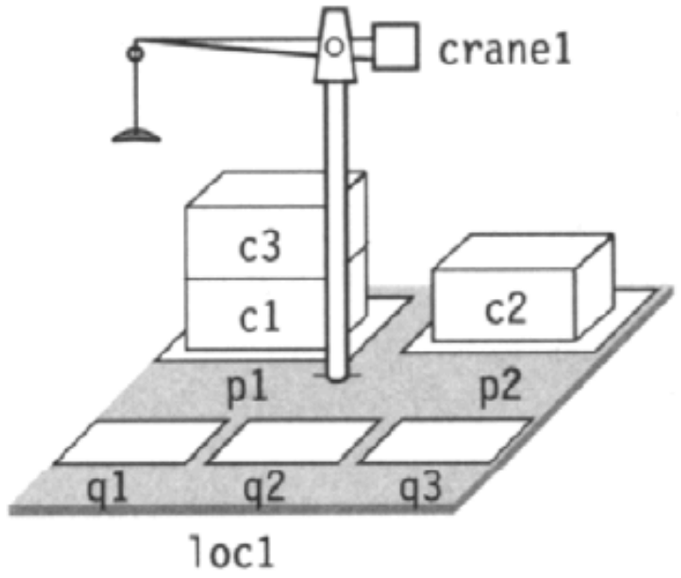
- The Sussman Anomaly shows the limitations of non-interleaved planning methods
- Before this was described, people used to do planning by considering different subgoals in SEQUENCE
- The Anomaly will show that naively pursuing one subgoal X after you satisfy the other subgoal Y may not work because steps required to accomplish X might undo things subgoal Y

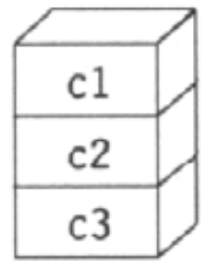
Descriptions of the world Σ, the initial state(s), and the objectives



The world Σ in which the agent operates



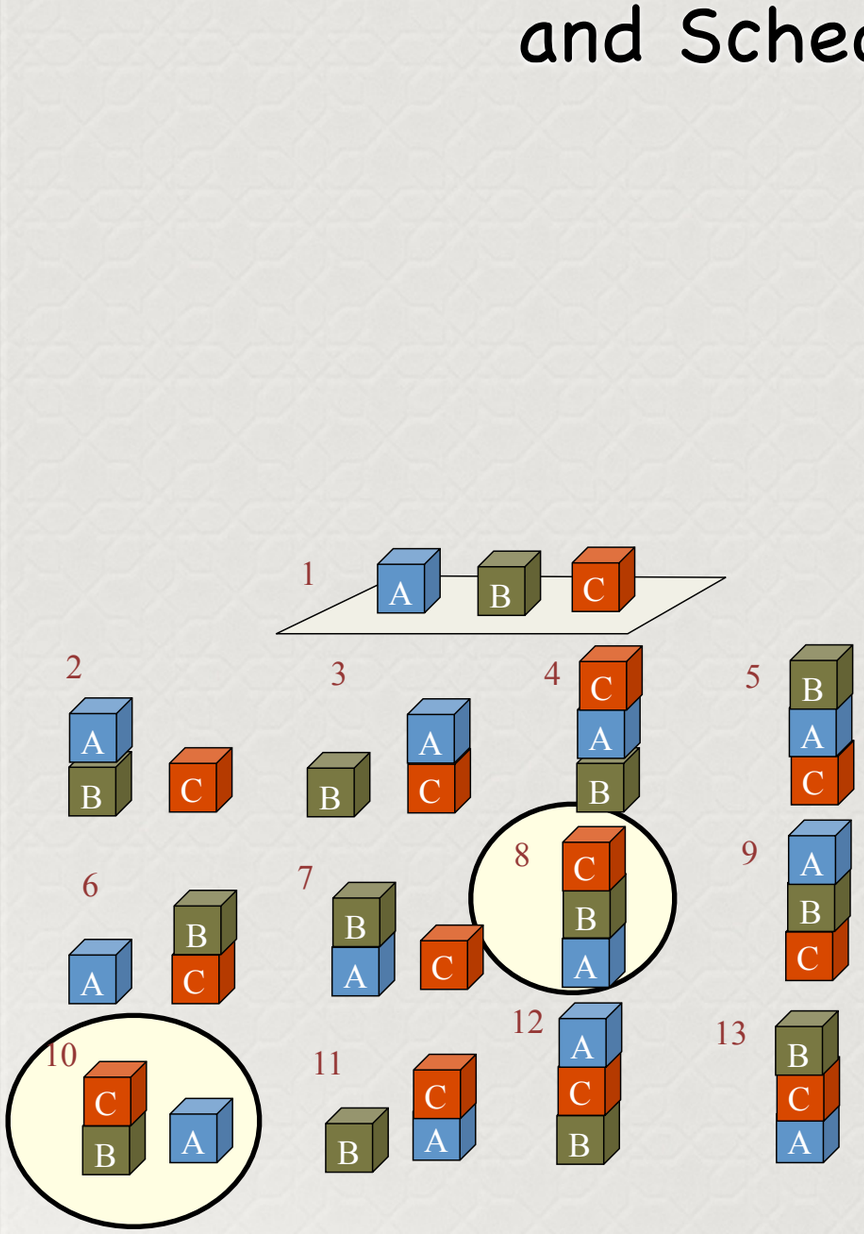


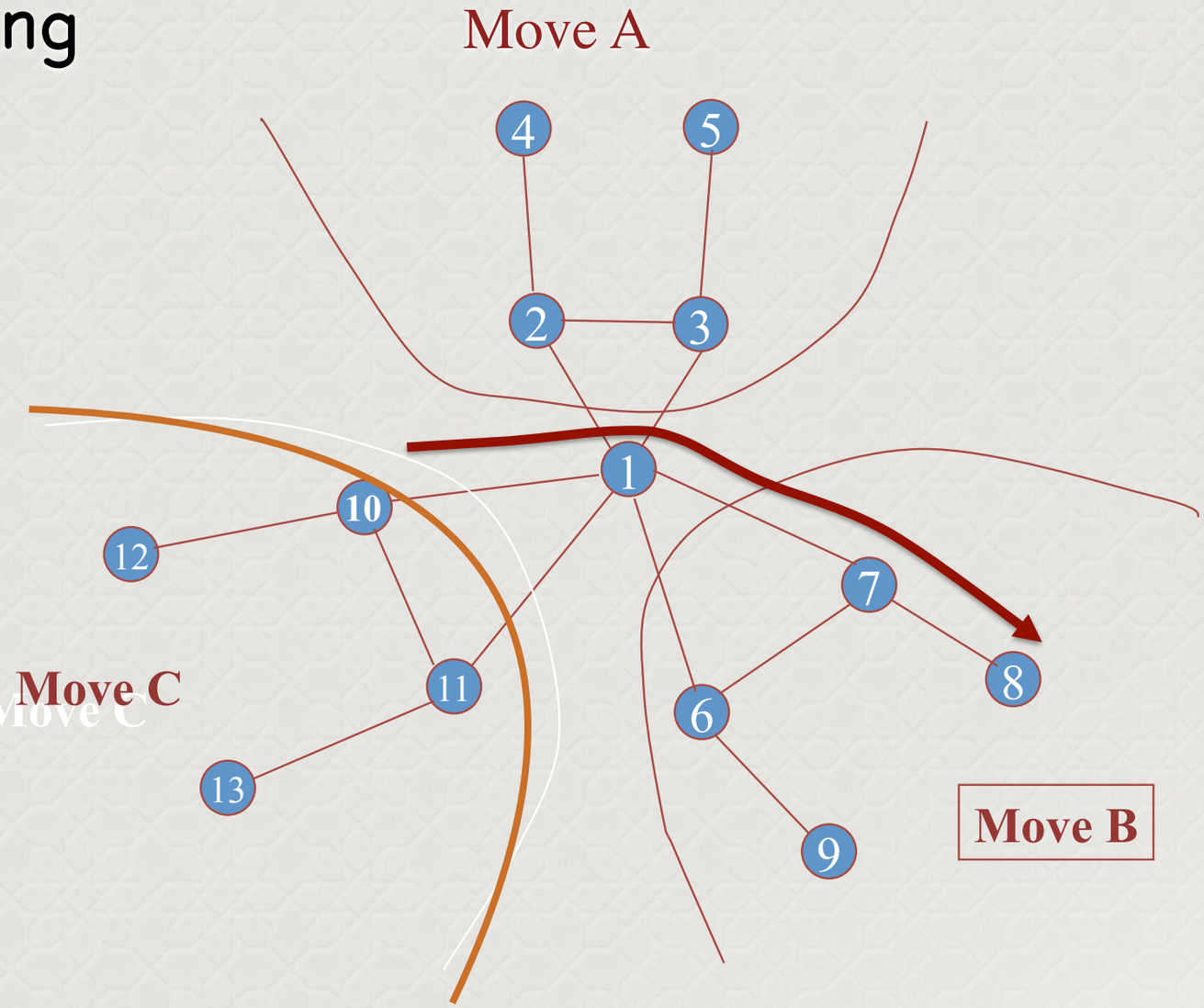


s_0={in(c3,p1), top(c3,p1), in(c1,p1), on(c3,c1)
on(c1,pallet), in(c2,p2), top(c2,p2),
on(c2,pallet), top(pallet,q1), top(pallet,q2)
top(pallet,q3), empty(crane1)}

g={on(c1,c2)
on(c2,c3)}

KEPS-Knowledge Engineering for Planning and Scheduling



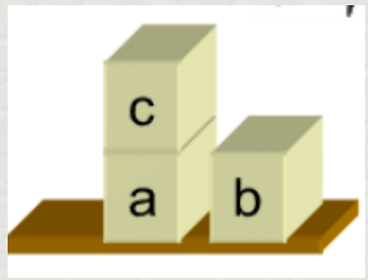


Exercício programa (Learn by doing)

Planner

Execution agent

Execution Plant



O estado da planta de execução consiste de uma lista de proposições lógicas descrevendo a configuração dos 3 elementos (blocos).

$$[on(a,b), on(b,mesa), on(c,mesa), livre(a), livre(c)]$$

A dinâmica da planta de execução é dada por um único operador, $move(X, Y)$ que move o elemento X para cima do elemento Y.

$$move(c, mesa)\,.$$

remove([sobre(c,b)], s) add([sobre(c,mesa), livre(b)], s)

sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)

sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)

Existem vários implementações do mundo de blocos na internet e até no SWISH Prolog (lembrando que os exercícios feitos no SWISH são em princípio públicos). A exercício consiste portanto da escolha e modificação de um programa que não seja "naive", isto é, que não incorra na anomalia de Sussman.

O mundo de blocos - versão D-Lab

SWISH File Edit Examples Help — 1 users online — Search

```
go(Start, Goal) :-
    bloco_fora_lugar(Start,Goal,H), %A função bloco_fora_lugar é a heurística
    adiciona([[Start],H], [], Been_list), %Adiciona a configuração inicial à Been_list, que é
    %uma lista contendo todos os nós que já foram expandidos e seus respectivos caminhos.
    path(Start,Goal,Been_list).
```

```
    !.

path(Start,Goal, Been_list):- %Nessa chamada de path, ocorre a expansão dos nós
    primeiro_lista_lista(Estado_Atual,Been_list),%Encontra o nó atual do primeiro elemento da li
    solucoes(Estado_Atual,Solucao), %Encontra os possiveis candidatos para serem expandidos
    primeiro_lista(Caminho,Been_list), %Retorna o caminho até o nó que está sendo expandido atua
    del_first(Been_list,Been_list1), %Deleta o nó pai cujo filho será gerado a seguir
    expande_no(Solucao,Goal,Caminho,Been_list1,New_Been_list), %Função que escolhe qual é o filh
    path(Start,Goal,New_Been_list).%Chama a função de novo e continua verificando os estados até

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Funções Auxiliares %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Função que movimenta um bloco X para cima de um bloco Y
move_b(State,Next) :-
    encontra_topo_lista(Y,State),
    encontra_topo_lista(X,State),
    atom_string(X,_),
    atom_string(Y,_),
    X \== Y,
    deleta_lista(X,State,Next1),
```

?- Your query goes here ...

Examples History Solutions — table results Run!

https://reinaldoswi.dlab-usp.org/p/Mundo de blocos - Rework 2.0.pl#tabbed-tab-0

O mundo de blocos - versão D-Lab

SWISH File Edit Examples Help 1 users online Search

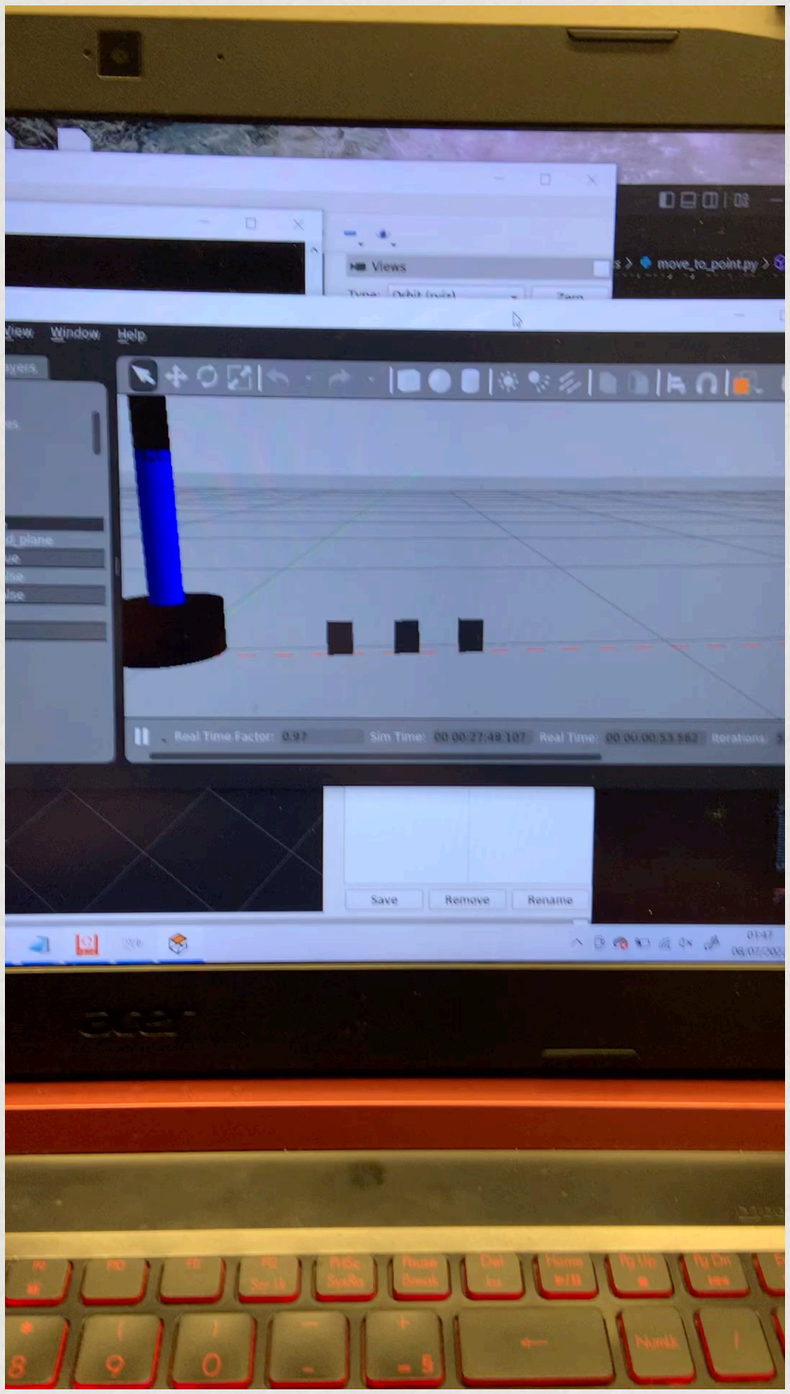
Mundo de b

```
path(_,Goal, Been_List):- %Nessa chamada de path, considera-se o caso em que o objetivo foi
    %encontrado
    primeiro_lista_lista(Objetivo,Been_List),
    Objetivo = Goal,
    primeiro_lista(Caminho,Been_List),
    inverte(Caminho,Caminho_I),
    write(Caminho_I),
    !.

path(Start,Goal, Been_list):- %Nessa chamada de path, ocorre a expansão dos nós
    primeiro_lista_lista(Estado_Atual,Been_list),%Encontra o nó atual do primeiro elemento da
    %lista
    solucoes(Estado_Atual,Solucao), %Encontra os possiveis candidatos para serem expandidos
    primeiro_lista(Caminho,Been_list), %Retorna o caminho até o nó que está sendo expandido
    %atualmente
    del_first(Been_list,Been_list1), %Deleta o nó pai cujo filho será gerado a seguir
    expande_no(Solucao,Goal,Caminho,Been_list1,New_Been_list), %Função que escolhe qual é o
    %filho mais promissor e coloca ele junto com seu caminho na lista contendo todos os nós
    path(Start,Goal,New_Been_list).%Chama a função de novo e continua verificando os estados
    %até chegar ao objetivo
```

https://reinaldos

```
30
31 %Função que movimenta um bloco X para cima de um bloco Y
32 move_b(State,Next) :-
33     encontra_topo_lista(Y,State),
34     encontra_topo_lista(X,State),
35     atom_string(X,_),
36     atom_string(Y,_),
37     X \== Y,
38     deleta_lista(X,State,Next1),
39     insere_lista(X,Y,Next1,Next).
40
41 %Função que movimenta um bloco X para a mesa
42 move_table_b(State,Next) :-
43     encontra_topo_lista(X,State),
44     atom_string(X,_),
45     vazia_lista(State),
46     deleta_lista(X,State,Next1),
47     insere_vazia_lista(X,Next1,Next),
48     Next \= State.
```


Views
move_to_point.py
Window
Help
Real Time Factor: 0.97
Sim Time: 00 00:27:48.107
Real Time: 00 00:00:53.562
Iterations:
Save
Remove
Rename
acer

Na próxima aula continuaremos a discussão sobre estados preferenciais em um processo de planning e possíveis heurísticas para melhorar a performance dos planejadores.

Perguntas?